The R language

Martin Frigaard

2021-04-28

Contents

R is an object-oriented, functional programming language for statistical analysis and graphics. R is also a free and open-source software (FOSS) with a massive global community of users and developers who have helped create and maintain tools for data manipulation, graphics, statistics, and machine learning.

0.1 Functions and objects

The R language is comprised of functions and objects. R uses functions to perform operations (like `mean()`, `sum()`, `lm()` (for linear model)) on objects (vectors, arrays, matrices, data.frames or lists).

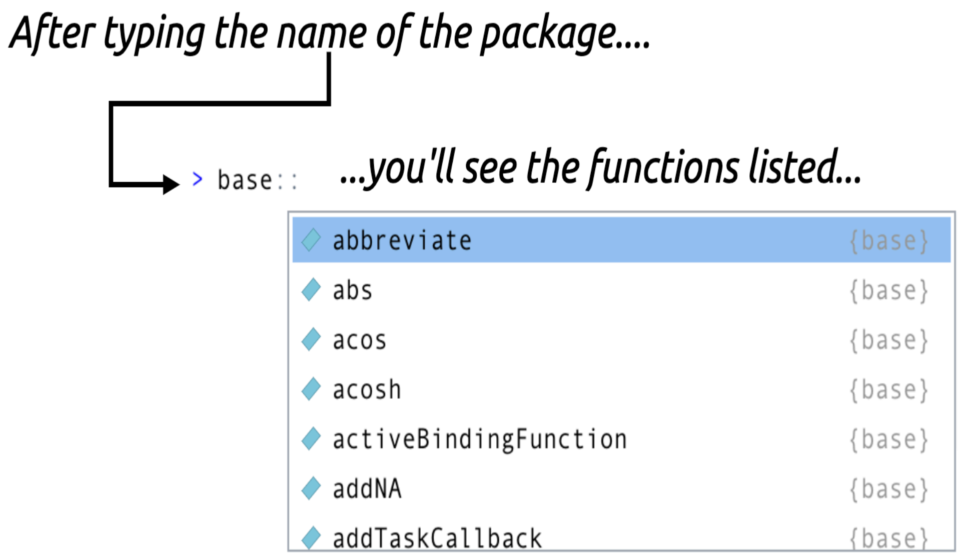
Generally speaking, functions are similar to *verbs*, and objects are more like *nouns*. Functions typically take an object as an `input`, perform an operation on that object, and then return an `output` object.

```
object <- function('input') {

    perform operation(s) on 'input'

    return output
}
# view object
object
```

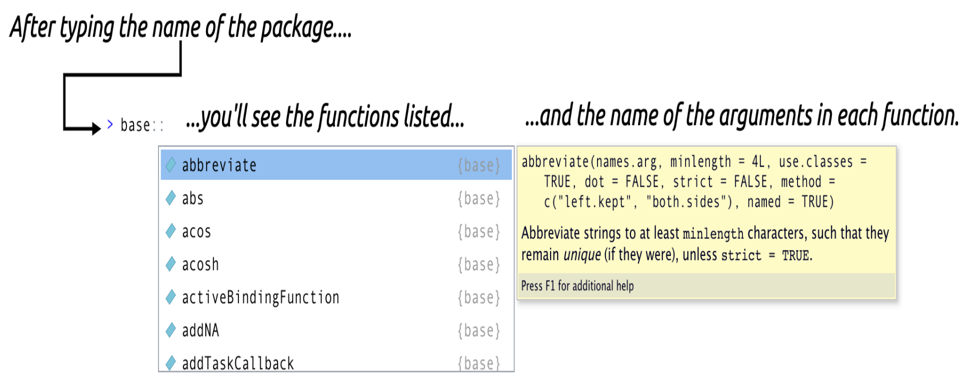
0.2 Base R packages

R `packages` are collections of commands for a particular purpose or task. R comes 'out of the box' with a ton of functions for manipulating, analyzing, and visualizing data. Two of the most commonly used standard packages are `base` and `utils`.

You can access any function in a package using the `package::function()` syntax. If you're in RStudio, you can actually see the functions in each package by using the tab-completion feature:


After typing the name of the package....
> base::
...you'll see the functions listed...
abbreviate {base}
abs {base}
acos {base}
acosh {base}
activeBindingFunction {base}
addNA {base}
addTaskCallback {base}

If you hover over the function with your mouse cursor, you'll also see the arguments and documentation for each function.


After typing the name of the package....
> base::
...you'll see the functions listed...
...and the name of the arguments in each function.
abbreviate {base}
abs {base}
acos {base}
acosh {base}
activeBindingFunction {base}
addNA {base}
addTaskCallback {base}

```
abbreviate(names.arg, minlength = 4L, use.classes =
    TRUE, dot = FALSE, strict = FALSE, method =
    c("left.kept", "both.sides"), named = TRUE)
```

Abbreviate strings to at least `minlength` characters, such that they remain *unique* (if they were), unless `strict = TRUE`.

Press F1 for additional help

When you're using the `utils::install.packages()` function, the package files are installed from the Comprehensive R Archive Network, or CRAN. These packages have passed a variety of tests and are generally considered to be more reliable.

0.3 User-written packages

Most of the packages we'll be using in this course come from the `tidyverse`, which is a suite of tools pioneered by RStudio's Chief Scientist Hadley Wickham. All packages in the `tidyverse` work well together because they center around a common thread of tidy data.

To install and load the `tidyverse`, we will use the `utils::install.packages()` function to download and installs R packages into a local folder on our computer, and the `base::library()` command loads the packages.

```
install.packages("tidyverse")
library(tidyverse)
```

NOTE: Not all functions return an output. Some functions return messages (or prompts), so be sure to check the help files by using `?install.packages` in the console.

User-written packages can be installed from code repositories like Github.

First you will need to install the `remotes` package from CRAN, then you can use the `remotes::install_github()`

```
install.packages("remotes")
library(remotes)
```

Now we can use the `remotes::install_github()` to download and install the `tidyverse` package.

```
remotes::install_github("tidyverse/tidyverse")
library(tidyverse)
```

Note: when installing packages from Github or other repos, you're getting the 'freshest' version, so there might be bugs or errors. If you run into an issue, look for a version of the package on CRAN

0.4 Creating objects

R comes with a variety of functions for creating objects. We will start with `c()`, which stands for 'combine' or 'concatenate'.

We can print this to the console by supplying the new object and hitting enter/return.

```
x <- c(42, 34, 28, 53, 71, 30, 23, 72, 59, 46, 64, 33, 42, 50, 68)
x
```

```
##  [1] 42 34 28 53 71 30 23 72 59 46 64 33 42 50 68
```

A quick note on printing: notice the preceding `[1]` in the output. This is not part of the object, it's the line number for the output.

Now that we have an object in R, what do we do with it? We will start by taking a look at some of it's technical information using `class()` and `str()`

```
class(x)
```

```
## [1] "numeric"
```

The `class()` function tells us `x` is a `numeric` vector. The `str()` function is an abbreviation for 'structure', and it gives us a bit more information.

```
str(x)
```

```
##  num [1:15] 42 34 28 53 71 30 23 72 59 46 ...
```

I recommend using `str()` and `class()` when you're programming in R. Knowing what kind of object you're dealing with will help you determine what you can do with it.

0.5 Store and explore

Given the relationship between functions and objects, a common workflow is to create (or import) data as an object (`my_data`), apply a function to this object (`log10()`), store the output from this function in a new object (`my_result`), and then use other functions to explore the `my_result`:

```
# create data
my_data <- c(49, 147, 74, 90, 7, -79, 190, 49, -123, -325, 143, 232)
# apply a function and store in my_result
my_result <- sqrt(my_data)
# explore the result
summary(my_result)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   2.646   7.000   9.487   9.759  12.124  15.232       3
```

Operators are symbols (or collections of symbols) for performing arithmetic (`+`, `-`, `*`, `/`), comparisons (`<`, `>`, `=<`, `=>`), and assignment (`<-` and `=`). *Operators* are also functions (see below).

```
class(`+`)
```

```
## [1] "function"
```

```
class(`<=`)
```

```
## [1] "function"
```

```
class(`<-`)
```

```
## [1] "function"
```

0.6 Data in R

When we load data into R, these data get stored in a data object. To do things to any data object (manipulate, analyze, visualize, model, etc.), we'll need to use functions. We can write custom functions, or we can use one of the over 10,000 user-written packages available on CRAN.

0.7 R Code style

The code in this scenario follows the `tidyverse` style guide as closely as possible.

> "*Each line of a comment should begin with the comment symbol and a single space:* `#`"

```
# comments aren't run in the console
```

> "*In data analysis code, use comments to record important findings and analysis decisions. If you need comments to explain what your code is doing, consider rewriting your code to be clearer.*"

0.8 Using R functions

If we want to use a function from a package, the syntax for doing this is `package::function()`

For example, below, we'll use the `tidyverse_logo()` function from the `tidyverse` package to view an awesome logo.

```
# click to execute code
tidyverse::tidyverse_logo()
```

```
##   __  _    __   .                  .
##  / /_(_)__/ /_ ___  ____ _______ ___
## / __/ / _  / // / |/ / -_) __(_-</ -_)
## \__/_/\_,_/\_, /|___/\__/_/ /___/\__/
##         . /___/             .
```

The `tidyverse::tidyverse_logo()` function can run without any arguments (i.e. nothing inside the parentheses), but we can view the arguments by placing the cursor inside the parenthesis and hitting the tab key.

We can enter function arguments by position or name (see below).

1. Tacutu, R., Craig, T., Budovsky, A., Wuttke, D., Lehmann, G., Taranukha, D., Costa, J., Fraifeld, V. E., de Magalhaes, J. P. (2013) "Human Ageing Genomic Resources: Integrated databases and tools for the biology and genetics of ageing." Nucleic Acids Research 41(D1):D1027-D1033.